ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Ф. В. Филиппов

ОБРАБОТКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В ФОРМАТЕ SVG

УЧЕБНОЕ ПОСОБИЕ

Часть 2

САНКТ-ПЕТЕРБУРГ
2017

УДК 004.92(075.8)
ББК 32.973-018.2я73
Ф 53

Рецензенты:
кандидат технических наук, доцент кафедры робототехники
и автоматизации производственных систем СПбГЭТУ «ЛЭТИ»
А. В. Шевченко,
кандидат технических наук, доцент кафедры конструирования
и производства радиоэлектронных средств СПбГУТ
Т. В. Матюхина

Утверждено редакционно-издательским советом СПбГУТ
в качестве учебного пособия

Филиппов, Ф. В.
Ф 53 Обработка графической информации в формате SVG : учебное пособие : часть 2 / Ф. В. Филиппов ; СПбГУТ. – СПб., 2017. – 36 с.

Рассматриваются практические аспекты использования формата SVG при разработке структурированных графических компонентов информационных управляющих систем на базе web-технологий.

Предназначено для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии», и будет полезно при изучении дисциплин «Технология программирования», «Технологии обработки информации» и «Технологии проектирования программного обеспечения информационных систем».

УДК 004.92(075.8)
ББК 32.973-018.2я73

Содержание

4. ИНТЕРАКТИВНОСТЬ *SVG*

Интерактивность *SVG*-документа обеспечивается наличием достаточно развитых средств для ее реализации. В первую очередь это гиперссылки для связывания между собой как различных ресурсов внутри документов, так и внешних страниц и файлов. Немаловажными также являются средства реагирования на внешние события – мышь, работа с документом, состояния процесса анимации. Интерактивность обеспечивается также благодаря абсолютной способности взаимодействия *SVG* и *HTML*. Внутрь *HTML*-страницы легко встраивается код *SVG*-документа или целиком подключается внешний *SVG*-файл. Можно, наоборот, внутри *SVG*-файла разместить код *HTML*.

Наконец, огромную роль играет возможность использования сценариев на базе языка *JavaScript* и широкого спектра его библиотек.

4.1. Гиперссылки

Одна из наиболее распространенных форм интерактивности – это переходы по гиперссылкам. Так же, как и в *HTML*, для создания ссылки используется тег *<a>*. Для указания адреса ссылки используется атрибут *xlink:href*.

Переход по ссылке выполняется по нажатии клавиши мыши, если курсор наведен на ссылочный объект. Этим объектом может быть просто слово, текст или любое изображение. Все, что относится к внешнему виду и поведению ссылочного элемента, содержится внутри тега *<a>*. В табл. 4.1 приведен пример простейшей гиперссылки, оформленной в виде оранжевого прямоугольника с закругленными краями и надписью белого цвета. Естественно, надпись подразумевает тот ресурс, который станет доступным по ссылке, в данном случае – это сайт университета.

Таблица 4.1

Пример использования тега *<a>* для создания гиперссылки

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <a xlink:href="https://www.sut.ru" target="_blank"> <rect height="30" width="100" y="10" x="10" rx="15" fill="darkorange"/> <text fill="white" x="18" y="31">БОНЧ здесь!</text> </svg>	

По умолчанию ссылка всегда открывается в родительском окне, а для того чтобы она открывалась в новом окне, надо внутри тега <*a*> добавить атрибут *target* или *xlink:show*, со значением *target ="new"* или *xlink:show ="new"*. Если необходимо, чтобы при наведении мыши на ссылочный элемент появлялся комментарий, то следует внутри тега <*a*> добавить атрибут *xlink:title,* его значение и будет появляться в качестве комментария.

В листинге 4.1 приведен пример ссылки в виде текста «http:// example.com/link (Откроется в новом окне)» , которая открывается в новом окне. При наведении курсора мыши на ссылку появляется комментарий «Переход к иллюстративному домену».

Листинг 4.1. Гиперссылка с комментарием, открывающаяся в новом окне:

```
<svg xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">
 <a xlink:href="http://example.com/link/"
    xlink:title="Переход к иллюстративному домену"
    xlink:show="new">
      <text x="10" y="15">http://example.com/link/
       (Откроется в новом окне)</text>
   </a>
</svg>
```

Домен http://example.com/link/ установлен для использования в иллюстративных примерах, приведенных в документах. Можно использовать этот домен в примерах без предварительного согласования или разрешения.

В табл. 4.2 приведен пример ссылки, оформленной в виде небольшого изображения земного шара с надписью. При переходе по ссылке в новом окне открывается изображение земли в увеличенном формате.

Таблица 4.2

Пример использования тега <*a*> для создания гиперссылки

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <a xlink:href="earth.png" target="new"> <image x="10" y="10" width="32" height="32" xlink:href="earth.png"/> <text x="2" y="60"> Земля</text> </svg>	

Поскольку ссылочным элементом может быть любой объект *SVG*, то к нему также применимы теги и атрибуты анимации. В качестве примера в коде табл. 4.2 анимируем ссылочный элемент – заставим земной шар

вращаться. Для этого в тег <*image*> добавим тег <*animateTransform*> с соответствующими атрибутами. В результате получим код листинга 4.2.

Листинг 4.2. Пример анимированной гиперссылкой:

```
<svg xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink">
 <a xlink:href="http://example.com/link/">
 <image x="10" y="10" width="32" height="32" xlink:href="earth.png">
 <animateTransform attributeName="transform" attributeType="XML"
   type="rotate" from="0, 26, 26" to="360, 26, 26"
   begin="0" dur="10s" repeatCount="indefinite" />
 </image>
 <text x="2" y="60"> Земля</text>
</a>
</svg>
```

Внешний вид ссылочного элемента здесь не изменился по сравнению с примером из табл. 4.2, за исключением того, что земной шар начал вращаться.

4.2. События

Спецификация консорциума *W3C* разбивает атрибуты, поддерживающие события по трем типам: атрибуты событий для объектов, атрибуты событий *SVG*-документа и атрибуты событий анимации.

Наиболее привычными являются события мыши такие, как *click*, *mousemove*, *mouseover*, *mouseout*, *mousedown* и *mouseup*. Кроме того, сюда можно отнести события связанные с попаданием объекта в «фокус» (*focusin*) и выхода из него (*focusout*), поскольку *click* над объектом делает его в «фокусе», а *click* вне его «расфокусирует» объект. В табл. 4.3 приводятся примеры применения некоторых событий мыши.

Поясним работу приведенного *SVG*-кода.

Красный круг реагирует на событие перемещения курсора мыши *begin="mousemove"* по его поверхности – он меняет цвет с красного на зеленый. Как только курсор мыши выходит за пределы круга *end="mouseout"*, он снова становится красным.

Желтый круг по нажатии и отпускании клавиши мыши *begin="click"* за три секунды *dur="3s"* увеличит свой радиус до ста пикселов *to="100"* и возвратится в исходные размеры.

Зеленый круг по нажатии и отпускании клавиши мыши станет в «фокусе» *begin="focusin"* (он будет обведен в голубую рамку), за три секунды *dur="3s"* переместится вверх, до координаты *cy = 50* и мгновенно вернется

на место. Заставить зеленый круг повторить предыдущее действие можно будет только тогда, когда он снова окажется в «фокусе». Но для этого он должен оказаться сначала не в фокусе (рамка должна исчезнуть). Это можно сделать, нажав и отпустив клавишу мыши, когда курсор вне зеленого круга. Рамка исчезнет, и можно будет все повторить сначала.

Таблица 4.3

Пример обработки событий мыши

Код SVG	Рисунок SVG
<svg xmlns="http://www.w3.org/2000/svg"> <circle cx="50" cy="50" r="40" fill="red" stroke="black" stroke-width="2" > <set attributeName="fill" to="green" begin="mousemove" end="mouseout"/> </circle> <circle cx="50" cy="150" r="40" fill="yellow" stroke="black" stroke-width="2"> <animate attributeName="r" from="40" to="100" begin="click" dur="3s"/> </circle> <circle cx="50" cy="250" r="40" fill="green" stroke="black" stroke-width="2"> <animate attributeName="cy" from="250" to="50" begin="focusin" dur="3s" end="focusout"/> </circle> </svg>	

4.3. Взаимодействие с HTML

Формат и документы *SVG* имеют самое тесное взаимодействие с форматом и документами *HTML*. Это не удивительно потому, что у них одна основа – язык *XML*. Как любой *SVG*-объект может быть легко встроен в любое место на страницу *HTML*-документа, так и наоборот, *HTML*-код может быть использован внутри *SVG*-документа.

Для встраивания *SVG* в *HTML* существует много способов, например, можно описывать *SVG* непосредственно в *HTML (inline)* или использовать:

- тег <*object*>;
- тег <*iframe*>;
- тег <*img*>;
- тег <*embed*>;
- *CSS Background Image.*

На практике использование того или иного способа зависит как от вкусов дизайнера, так и от назначения проектируемого веб-ресурса. В любом случае важно учитывать, какие манипуляции предполагается применять к контенту: *CSS*-манипуляции, *JavaScript*-манипуляции, *SVG*-анимации и интерактивные *SVG*-анимации.

Все перечисленные виды манипуляций будут поддерживаться при использовании двух первых способов встраивания – *inline* и с помощью тега <*object*>. Понятно, что непосредственное встраивание кода *SVG* весьма неудобно, особенно при громоздких и сложных проектах. Именно поэтому следует считать оптимальным со всех точек зрения использование тега <*object*>, как самый надежный метод для обеспечения интерактивности:

```
<object type="image/svg+xml" data="image.svg" width="50" height="25">
    Ваш браузер не может воспроизвести это изображение!
</object>.
```

В случаях когда *HTML*-файл будет обрабатываться браузером, который не может по каким-либо причинам воспроизвести указанный *SVG*-файл, появится данное сообщение.

Поскольку все браузеры способны так или иначе справляться с интерпретацией *SVG*-файлов, можно использовать тег <*iframe*>:

```
<iframe src="image.svg">
    <img src="image.png" width="50" height="25” alt=”image format png” />
</iframe>
```

Здесь для «плохих» браузеров предложено изображение в формате *PNG*, которым они будут заменять непонятный им файл *image.svg.*

Тег <*img*> можно использовать, когда *SVG*-файл играет роль обычной картинки, поскольку возможности преобразований в этом формате ограничены. Аналогичное замечание справедливо и для *CSS Background Image.* Использование тега <*img*> для обслуживания «плохих» браузеров может осуществляться следующим образом:

```
<img src="image.svg" width="50" height="25"onerror="this.onerror=null;
            this.src=''image.png">
```

Замена на файл image.png здесь будет происходить, когда браузер получит ошибку при обработке *SVG*-файла.

Тег <*embed*> записывается в формате:

```
<embed type="image/svg+xml" src="image.svg" />
```

Несмотря на то что тег <*embed*> поддерживается большинством браузеров, он не является частью спецификации *HTML*, поэтому имеет ограниченное применение.

В *SVG*-документы можно включать внешние элементы, имеющие иное пространство имен. В частности, есть возможность вставки *HTML*-кода, которая оказывается весьма кстати при внедрении видео, веб-страниц и тому подобного внутри *SVG*-документа.

Для вставки внешних объектов применяется контейнерный тег <*foreignObject*>. Прямоугольная область, в которой отображается содержимое внешнего объекта, задается, как обычно, атрибутами: *x*, *y* – координаты верхнего левого угла, *width* и *height* – ширина и высота области.

В табл. 4.4 приведен пример вставки в SVG-документ трех фреймов, в которые в качестве контента загружается содержимое трех сайтов.

Таблица 4.4

Пример вставки *HTML* в *SVG*

Код *SVG*	Рисунок *SVG*
<svg xmlns = "http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="800" height="800"> <foreignObject x="80" y="30" width="800" height="800"> Комиксы на XKCD: <iframe src="http://xkcd.com/559/" width="300" height="100"></iframe> ОБЗОР ВОЗМОЖНОСТЕЙ OWL1 и OWL2: <iframe src="http://trinidata.ru/tech_owl.htm" width="300" height="100"></iframe> Газета RU! <iframe src="http://gazeta.ru" width="300" height="100"></iframe> </foreignObject> </svg>	

Аналогичным образом в *SVG*-документ можно вставить и другие внешние объекты, которыми могут быть, например, звуковые и видеофайлы. Более подробную информацию по встраиванию различных внешних объектов в SVG можно получить в спецификации [2].

4.4. Сценарии JavaScript

Сценарии *JavaScript* при работе с *SVG* используют в различных целях: для создания и модификация объектов, анимации и обеспечения интерактивности документа.

Добавление скриптов внутрь *SVG* можно реализовывать как внутри документа, так и с помощью внешних ссылок. Внутрь документа *JavaScript* код вставляется с использованием следующей конструкции:

```
<svg xmlns="http://www.w3.org/2000/svg">
<script>
    <![CDATA[
      ...код на JavaScript...
    ]]>
</script>
</svg>
```

Ссылка на внешние скрипты реализуются с помощью *xlink*:

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink">
<script xlink:href="external.js" />
</svg>
```

Программа на *JavaScript* работает точно так же, как и в *HTML*-документе. Можно ссылаться на уникальные идентификаторы *id* объектов. Нужно только учитывать, что при создании и поиске идентификаторов объектов учитывается регистр.

Рассмотрим небольшой пример, сочетающий в себе использование сценариев *JavaScript* и взаимодействие с *HTML*.

Спроектируем простую игру, цель которой попасть по движущейся мишени. Каждое попадание должно быть зафиксировано в таблице результатов отметкой о координате *X* точки попадания.

Сначала сформируем мишень и таблицу результатов (листинг 4.3). Для движения мишени используем *SVG*-анимацию атрибута *cx* – центра мишени. Начальная строка в таблице, сформированной тегом *HTML* <*ul*>, содержит запись «Результат:».

Листинг 4.3. Формирование движущейся мишени и таблицы результатов:

```
<html>
<h1>Попади в мишень </h1>
<svg height="150" width="100%">
  <circle id="aim" r="50" cy="75" fill="blue">
    <animate attributeName="cx"
          values="0%;100%;30%;65%;5%;95%;0%;75%;20%;90%;0%"
          dur="10s"
          repeatDur="indefinite"
          calcMode="paced" />
  </circle>
</svg>
<!--Формирование таблицы для записи результатов-->
<ul id="log"> Результат:</ul>
<!--Вызов внешнего JavaScript сценария-->
<script src="Hit.js"></script>
</html>
```

Фиксация попаданий будет реализована в виде внешнего сценария *Hit.js*, который вызывается из *HTML*-файла с помощью <*script src="Hit.js"*> </*script*>. Код внешнего сценария приведен в листинге 4.4.

Листинг 4.4. Фиксация попаданий по мишени:

```
var log = document.getElementById("log");
document.getElementById("aim")
      .addEventListener("mousedown", record);
var i = 0;
```

```
function record(opp){
  var c = opp.target;
     l = document.createElement("li");
  i++;
  l.textContent = [i,"."," Попадание в точке X = ",
                parseInt(c.cx.animVal.value),
                " пикс"
                ].join("");
  log.insertBefore(l, null);
}
```

Обращение к таблице записей результатов и мишени осуществляется с помощью их идентификаторов log и aim. Фиксация результата в таблице происходит только тогда, когда кнопка мыши была нажата точно над мишенью: *addEventListener("mousedown", record)*. Функция записи результата *record* сначала формирует новую строку в таблице *l = document.create Element ("li")*, а затем записывает результат в виде фразы: «1. Попадание в точке $X = 323$ пикс».

В приведенном сценарии *JavaScript* использовался для того, чтобы реализовать запись результата в таблицу, что невозможно реализовать силами *SVG*. Как показывает практика, часто используют сценарии *JavaScript* там, где можно обойтись силами *SVG*, но с помощью сценария это сделать гораздо проще. Особенно это свойственно программной анимации [38].

Программная анимация – это изменение параметра объекта по определенному математическому закону в зависимости от времени. Это может быть линейная функция, квадратичная функция, синусоида, асимптотическая функция и т. д. Для создания анимации нужно знать начальное и конечное значения анимируемого параметра, начальное время анимации и длительность анимации (в миллисекундах или кадрах). Например, если речь идет о программной анимации измеряемого параметра некоторого *SVG*-объекта с идентификатором *id="obj"*, то можно предложить универсальный сценарий (приведенные конкретные значения являются условными):

```
var element = document.getElementById("obj");
var from = 0; // Начальное значение параметра
var to = 500; // Конечное значение параметра
var duration = 1000; // Длительность – 1 секунда
var start = new Date().getTime(); // Время старта
setTimeout(function() {
   var now = (new Date().getTime()) – start; // Текущее время
   var go = now / duration; // Прогресс анимации
   var result = (to – from) * delta(go) + from;
   element.style.left = result + 5;
   if (progress < 1) // Если анимация не закончилась, продолжаем
      setTimeout(arguments.callee, 10);
}, 10);
```

Переменная *go* по сути определяет, какая часть сценария выполнена как отношение текущего времени *now* к длительности анимации *duration*. Текущее значение анимируемого параметра *result* можно заставить изменяться по любому закону *f* в зависимости от переменной *go*: *result = (to – from) * f(go) + from.* Например, в табл. 4.5 приведены некоторые примеры реализации функции *f*, определяющей закон изменения анимируемого параметра.

Таблица 4.5

Примеры реализации функции изменения анимируемого параметра

Функция *f*	*JavaScript*-реализация
$f = go^n$	function f(go) { return Math.pow(go, 2); }
f = 1-sin(arccos(go))	function f(go) { return Math.sin(Math.acos(go));}
$f = 2^{(10 * (go - 1))}*\cos(20*go*\pi*x/3)$	function f(go, x) { return Math.pow(2,10*(go-1))*Math.cos(20*go*Math.PI *x/3); }

Поскольку анимировать можно практически любой параметр и может быть выбрана любая функция *f*, то программная анимация с помощью *JavaScript* значительно упрощается. Попробуйте реализовать несколько анимаций, используя предложенный подход и выбирая в качестве параметра прозрачность, высоту или ширину объекта.

4.5. Библиотеки JavaScript

Естественно, создавать реальные, сложные приложения с использованием *SVG* без помощи библиотек – дело чрезвычайно трудоемкое. Именно для того чтобы облегчить труд программистов и веб-дизайнеров, существует *CDNJS – Content Delivery Network JavaScript* – сеть доставки контента, включающего *JavaScript*-библиотеки различного назначения. На сайте [43] есть замечательный видеоролик о развитии сети *CDNJS*, начиная с февраля 2011 г. Достаточно зайти на сайт [44], чтобы узнать, сколько библиотек имеется в свободном онлайн-доступе: на начало декабря 2016 г. их насчитывалось 2629. Конечно, нас в первую очередь интересуют библиотеки поддержки *SVG*. На этом же сайте можно найти ссылку на нужную библиотеку и сайт информационной поддержки.

Остановимся на знакомстве с пятью библиотеками: *snap.svg*, *svg.js*, *d*3, *vivus* и *kute.js*. Каждая из них имеет свои особенности, а в целом они составляют мощный инструментарий для реализации различных эффектов, доступных в *SVG* с помощью *JavaScript*.

Для работы с любой из библиотек достаточно в столбце *Library* на сайте [44] выбрать нужную библиотеку, в столбце *Link* открыть меню

Copy, выбрать пункт *Copy Script Tag* и скопировать полученный тег. Затем вставить скопированный тег внутри тега *<body>* главного файла своего проекта, обычно это файл с именем *index.html*.

Например, перейдем на сайт [44] и введем для поиска ключевое слово svg. В результате поиска будет выведена вся информация о библиотеках JavaScript, причастных к формату SVG (рис. 4.1).

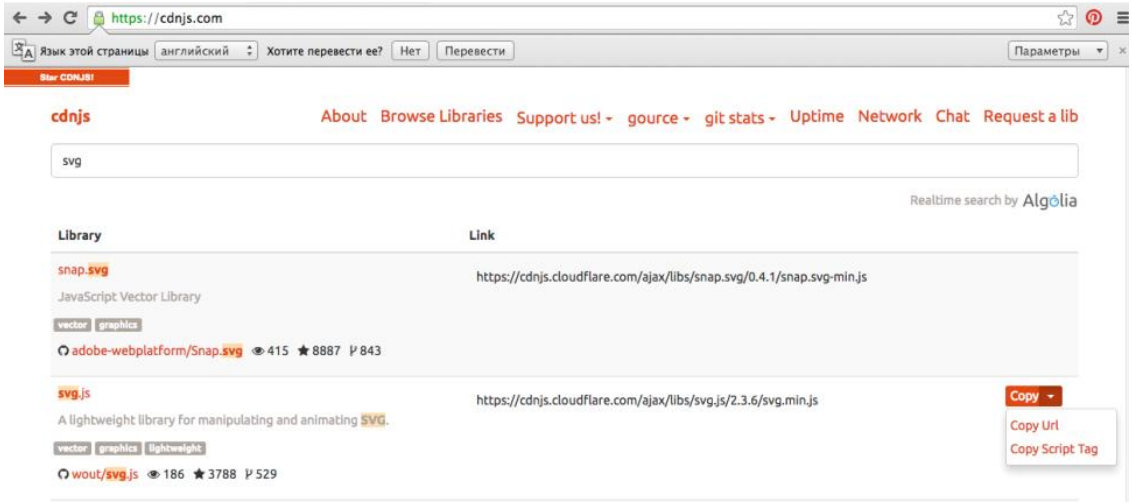


Рис. 4.1. Информация о библиотеках JavaScript, связанных с SVG

Допустим, требуется воспользоваться функциями библиотеки *svg.js*, для этого переходим к соответствующей строке таблицы, в столбце *Link* берем ссылку на эту библиотеку и формируем строку

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/svg.js/2.3.6/svg.min.js"></script>
```

в главном файле проекта. Для перехода на сайт информационной поддержки достаточно перейти по ссылке *wout/svg.js*, расположенной в левом нижнем углу строки таблицы.

Сеть доставки контента *CDNJS* обеспечивает доставку любой библиотеки в случае подключения к сети Интернет. Однако, если библиотека будет использоваться автономно, лучше скопировать ее в директорий проекта, тем самым обеспечив гарантию работы приложения при отключении от сети Интернет.

4.5.1. *Библиотека Snap.svg*

Snap.svg – это *JavaScript*-библиотека, которая является преемником библиотеки *Raphaël.js* [39] – одной из самых популярных библиотек для работы с форматом *SVG*. Эта новая библиотека включает изменения для поддержки большинства возможностей *SVG*, которые библиотека *Raphäel.js* не могла использовать. В частности, это такие функции, как *mask*, *group*,

gradient, *filter*, *animate* и *pattern*. Библиотека *Snap.svg* доступна под лицензией *Apache 2*, это означает, что ею можно пользоваться бесплатно, и она имеет полностью открытый исходный код.

Библиотека *Snap.svg* помогает создавать графику, но она также может работать с существующим *SVG*-объектом, полученным с помощью таких инструментов, как, например, *Adobe Illustrator* или *Inkscape*.

Для работы с библиотекой *Snap.svg* достаточно скачать ее из ресурса [40], разархивировать полученный файл, и в результате будут сформированы папки *demos*, *dist*, *doc*, *src* и *test.*

Содержимое папок библиотеки Snap.js:

1) *demos* – здесь находятся некоторые примеры использования библиотеки. В частности, папка tutorial с небольшим обучающим примером;

2) *dist* – содержатся уменьшенный (*snap.svg-min.js*) и несжатый (*snap.svg.js*) скрипты, используемые для разработки;

3) *doc* – здесь находится документация по *API*;

4) *src* – размещены компоненты, инструменты и плагины для работы Snap;

5) *test* – папка содержит модульные тесты для элементов библиотеки Snap.svg.

Новый веб-проект формируется в отдельном директории, который должен включать главный *HTML*-документ *index.html* и поддиректорий *js* с библиотекой *snap.svg.js* (взятой из *dist*) и рабочим файлом *main.js*. В этом случае шаблон *HTML* с подключенными скриптами будет выглядеть следующим образом (листинг 4.5):

Листинг 4.5. Главный HTML-документ index.html:

```
<html>
<head>
   <script src="js/snap.svg.js"></script>
  </head>
<body>
   <svg id="svg" width="800" height="500" viewBox="0 0 800 500"
    xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"></svg>
  <script src="js/main.js"></script>
 <svg id="svg"></svg>
</body>
</html>
```

Здесь внутрь <*body*> также вставлен контейнер <*svg*>, которому присвоен идентификатор *"svg"*.

В рабочем файле *main.js* (листинг 4.6) сначала инициализируем библиотеку указанием на контейнер <*svg*>, путем создания и назначения не-

которой переменной: *var s = Snap("#svg").* После чего через переменную *s* получаем доступ ко всем функциям и методам библиотеки Snap.svg.

Например, создадим фигуру в виде пирамидки с помощью пяти эллипсов (листинг 4.6).

Листинг 4.6. Рабочий файл main.js:

```
var s = Snap("#svg");
var e1 = s.ellipse(150,200,80,30);
var e2 = s.ellipse(150,180,60,20);
var e3 = s.ellipse(150,160,40,15);
var e4 = s.ellipse(150,140,20,10);
var e5 = s.ellipse(150,110,15,20);
```

При запуске главного документа *index.html* с кодом из листинга 4.6 в рабочем файле *main.js* получим результат в виде левой фигуры рис. 4.2. Как можно видеть, эллипсы по умолчанию имеют черный цвет заливки. Добавим в рабочий файл заливку:

```
// Раскрашиваем эллипсы
e1.attr({fill: 'mediumturquoise'});
e2.attr({fill: 'blue'});
e3.attr({fill: 'forestgreen'});
e4.attr({fill: 'yellow'});
e5.attr({fill: 'red'});
```

Получим результат в виде правой фигуры рис. 4.2.

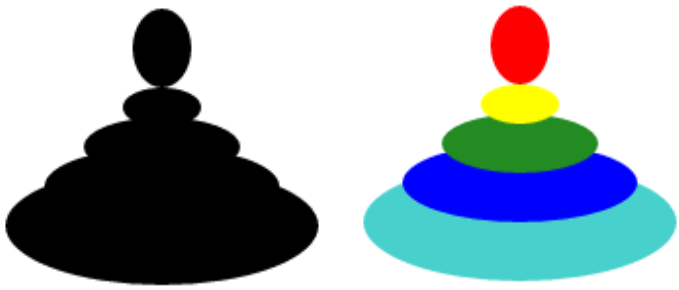


Рис. 4.2. Результаты выполнения скрипта *index.html* с рабочим файлом main.js

Рассмотрим некоторые особенности использования библиотеки *Snap.svg* на примере разработки простого приложения – назовем его моргающий глаз лягушки. Будем формировать рабочий файл *eye_frog.js* (листинг 4.7).

Листинг 4.7. Рабочий файл eye_frog.js:

```
var s = Snap("#svg");
var circle_1 = s.circle(300, 200, 140);
var circle_2 = s.circle(250, 200, 140);
var circle_3 = s.circle(275,200,80)
```

```
// Группируем круги вместе
var eye = s.group(circle_1, circle_2,circle_3);
var ellipse = s.ellipse(275, 220, 170, 80);
// Окрашиваем зрачок
circle_3.attr({fill:"black"});
// Окрашиваем контур глаза и задаем прозрачность
 ellipse.attr({fill: "white",opacity: 0.9});
// Окрашиваем радужку, задаем прозрачность
// и применяем маску
eye.attr({fill: "olive",fillOpacity: 0.6,mask: ellipse});
// Записываем функцию мигания
function wink(){
  ellipse.animate({ry:1}, 220, function(){
    ellipse.animate({ry: 90}, 275);}); };
// Вызываем функцию wink каждые 2,5 секунды
setInterval(wink, 2500);
```

Зрачок и радужную оболочку глаза сформируем из трех кругов, объединенных в группу *eye = s.group(circle_1, circle_2,circle_3),* а контур глаза формируем из эллипса: *ellipse = s.ellipse(275, 220, 170, 80).*

Закрашиваем зрачок в черный цвет, контур глаза делаем белым и почти полностью прозрачным: *ellipse.attr({fill: "white", opacity: 0.9}).* На этом этапе получается изображение, приведенное на рис. 4.3 слева.

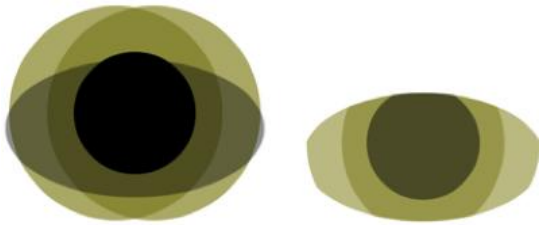


Рис. 4.3. Этапы формирования изображения глаза лягушки

Наконец, закрашиваем радужную оболочку в лягушачий цвет и маскируем контуром *eye.attr({fill: "olive",fillOpacity: 0.6,mask: ellipse}).* Глаз сформирован, он изображен на рис. 4.3 справа.

Теперь напишем функцию мигания *wink().* Глаз маскируется контуром в виде эллипса. Для того чтобы глаз открывался и закрывался, нужно изменять радиус эллипса по оси *Y* от 1 пиксела (*ry:1* полностью закрытый глаз) до, скажем, 90 пикселов (*ry:90*). Для этого используем функцию *animate.* Конечно, можно выбрать и другие пределы для *ry*. Можете поэкспериментировать.

Периодичность мигания можно установить по желанию, осуществляя вызов функции *wink()* с заданным интервалом, например *setInterval(wink, 2500),* 2,5 секунды.

Для тестирования работы программы *eye_frog.js* можно использовать в качестве главного *HTML*-документа программу *index.html* листинга 4.5, не забыв заменить имя рабочего файла *src="js/main.js"* на *src="js/ eye_frog.js".*

Большое число полезных и интересных примеров использования библиотеки *Snap.svg* можно найти на [41].

4.5.2. *Библиотека* svg.js

Библиотека svg.js предоставляет очень удобный инструментарий для реализации всевозможных трансформаций и анимаций *SVG* [35]. Она свободно распространяется на условиях *MIT*. Библиотека построена по модульному принципу, что позволяет оптимально распределять вычислительные ресурсы. Объем минимального ядра *svg.min.js* составляет всего 34 кбит, а отдельный модуль *svg.filter.js* занимает 3 кбит и является кросс-браузерным аналогом для свойства *webkit-filter*.

Следует отметить достаточно простой и понятный синтаксис [36] (листинг 4.8).

Листинг 4.8. Анимация перемещения и цвета прямоугольника:

```
<svg id="drawing" xmlns="http://www.w3.org/2000/svg"
                  xmlns:xlink="http://www.w3.org/1999/xlink">
 <script xlink:href="svg.min.js"></script>
 <script>
  <![CDATA[
   var draw = SVG('drawing')
   draw.circle(100).animate().fill('green').move(300,100)
  ]]>
 </script>
</svg>
```

В листинге 4.8 сначала с помощью ссылки подключается минимальное ядро библиотеки *svg.min.js*. Затем создается *SVG*-объект *draw*, в качестве которого выбирается круг с радиусом 100. Далее задается анимация заполнения цветом и перемещения в точку к координатами 300, 100.

Функциональные возможности библиотеки широки:

- анимация размеров, позиции, трансформации, цвета;
- полная поддержка масок;
- группировка элементов;
- динамические градиенты;
- привязка событий к элементам, включая события прикосновения (touch);
- text paths с поддержкой анимации.

Возможно также подключение различных модулей для реализации определенных дополнительных возможностей, например:

- svg.filter.js – набор SVG-фильтров;
- svg.draggable.js – возможность перетаскивания объектов;
- svg.easing.js – задание закона изменения параметра анимации;
- svg.path.js – создание кривых;
- vg.math.js – набор математических функций;
- svg.foreignobject.js – реализация foreignObject (объект SVG, внутрь которого вставляется произвольный HTML);

• svg.export.js и svg.import.js – импорт и экспорт SVG.

Приведем несколько простых примеров, поясняющих особенности использования библиотеки svg.js.

Рассмотрим использование текста в качестве маски для вырезания. Возьмем, например, изображение звездного неба stars.jpeg как фон для вырезания и текст «SVG.JS». Соответствующий код приведен в табл. 4.6.

Таблица 4.6

Маскирование изображения текстом

Код *JavaScript*	*SVG*-рисунок
<svg id="drawing" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"> <script xlink:href="svg.min.js"></script> <script> <![CDATA[var draw = SVG('drawing') var image = draw.image('stars.jpeg') image.size(650, 650).y(-200) var text = draw.text('SVG.JS').move(300, 0) text.font({family: 'Arial', size: 120, anchor: 'middle'}) image.clipWith(text)]]> </script> </svg>	

Следующий пример показывает возможности использования библиотеки *svg.js* для привязки объектов к событиям (листинг 4.9). В частности, в качестве объекта берется квадрат 200×200 с радиусом скругления углов 100, что превращает его в круг. Объект заставляем перемещаться 10 раз в точку с координатами 400, 300: *rect.animate().move(400,300).loop(10).* Если навести на объект курсор мыши, то он остановится *rect.mouseover(function() {this.pause()})*, а если отвести курсор мыши *rect.mouseout(function() {this.play()})*, то он продолжит движение.

Листинг 4.9. Обработка события наведения мыши на SVG объект:

```
<svg id="drawing" xmlns="http://www.w3.org/2000/svg"
            xmlns:xlink="http://www.w3.org/1999/xlink">
  <script xlink:href="svg.js"></script>
  <script>
    <![CDATA[
var draw = SVG('drawing')
var rect = draw.rect(200,200)
rect.radius(100)
rect.fill('yellow')
rect.animate().move(400, 300).loop(10)
rect.mouseout(function() {this.play()})
rect.mouseover(function() {this.pause()})
```

```
   ]]>
  </script>
</svg>
```

Предыдущие примеры использовали только минимальное ядро библиотеки. Покажем, как можно использовать дополнительные модули. Например, создадим объекты, которые можно свободно перемещать по всему холсту (листинг 4.10). Для этого добавим ссылку на дополнительный модуль *xlink:href="svg.draggable.js"*.

Листинг 4.10. Обеспечение возможности перетаскивания SVG объектов:

```
<svg id="drawing" xmlns="http://www.w3.org/2000/svg"
            xmlns:xlink="http://www.w3.org/1999/xlink">
  <script xlink:href="svg.min.js"></script>
  <script xlink:href="svg.draggable.js"></script>
  <script>
    <![CDATA[
var draw = SVG('drawing')
var c1 = draw.circle(100)
var c2 = draw.circle(100)
c1.fill('red')
c1.draggable()
c2.fill('blue')
c2.draggable()
    ]]>
  </script>
</svg>
```

В качестве объектов выберем красный и синий круги радиусом 100. Поскольку красный круг будет сформирован раньше, он окажется закрыт синим кругом. Делаем круги перемещаемыми: *c1.draggable()* и *c2.draggable()*. Теперь можно запустить скрипт листинга 4.10 и убедиться в возможности перетаскивания.

4.5.3. *Библиотека Vivus.js*

Библиотека *Vivus.js* специализируется на анимации процесса рисования объектов *SVG* [33]. Нечто подобное можно увидеть запустив код листинга 3.3. Для того чтобы получить этот эффект, в примере было использовано свойство *stroke-dasharray*. Это свойство позволяет управлять смещением обводки объектов *SVG*.

Библиотека *Vivus.js* использует для целей анимации *CSS* свойство *strokeDashoffset*. Поскольку свойство *strokeDashoffset* доступно только для объектов <path>, то анимация процесса рисования для таких объектов, как *<rect>*, *<circle>*, *<ellipse>*, *< polyline >* и *<polygon>*, напрямую не пройдет.

Для исправления этого недостатка следует дополнительно использовать специальный класс *pathformer*, который позволяет преобразовывать указанные объекты в объект <*path*>.

Анимация всегда рисует объекты в том же порядке, как они определены в теге *SVG*. При этом есть несколько условий, которым объекты *SVG* должны соответствовать, в частности:

- все объекты должны иметь свойство обводки, и она не может быть заполнена;
- текстовые объекты не могут использоваться для анимации.

При изучении возможностей и особенностей формирования кода для анимации рисования удобно использовать онлайн-ресурс [34]. На рис. 4.4 изображен интерфейс ресурса анимации рисования с использованием библиотеки *Vivus.js*.

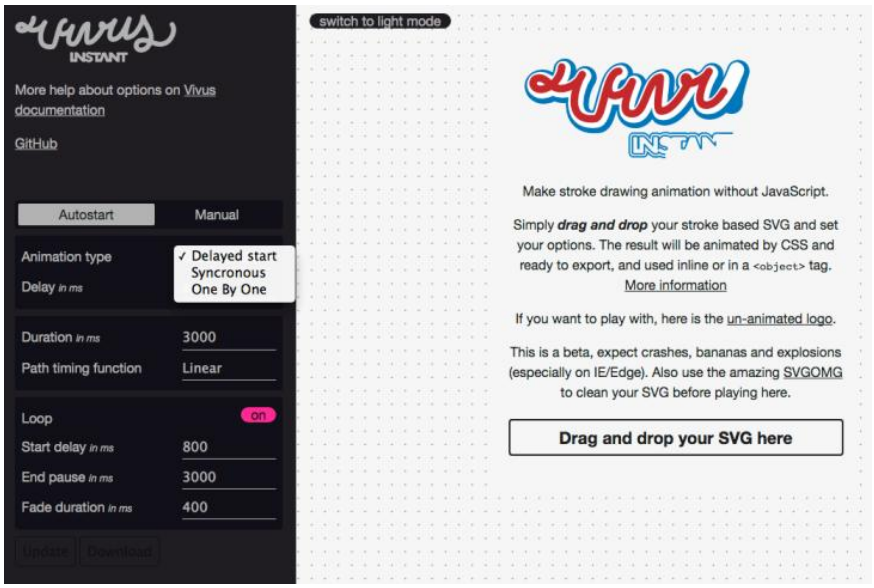


Рис. 4.4. Ресурс анимации рисования *Vivus.js*

Для использования этого ресурса достаточно просто перетащить *SVG*-объект на поле «*Drag and drop your SVG here*» и задать параметры анимации:

- тип анимации: задержанный старт (*Delayed Start*), одновременный старт (*Syncronous*) или друг за другом (*One By One*) – играет роль при анимации нескольких объектов;
- продолжительность анимации;
- способ изменения темпа;
- повторяемость.

Можно повторно изменять параметры анимации и отлаженный код выгрузить себе в компьютер для изучения и использования.

Для разработки *SVG*-документов саму библиотеку *Vivus.js* можно загрузить из ресурса *https://cdnjs.cloudflare.com/ajax/libs/vivus/0.4.0/vivus.js*.

4.5.4. *Библиотека d3.js*

Библиотека *d3.js* представляет собой набор из 30 модулей, которые предназначены для совместной работы. Их можно использовать независимо друг от друга или вместе как часть сборки по умолчанию. Источник и документация для каждого модуля доступны в репозитории [30]. Название *d3* образовано от *Data Driven Documents*, тем самым делается упор на то, что функции библиотеки ориентированы на обработку и визуализацию данных.

Библиотека *d3.js* основана прежде всего на использовании *JavaScript*, *SVG* и *CSS* в противовес другим подобным библиотекам, которые вместо *SVG* используют элемент *canvas* и его возможности.

Во многом благодаря этому *d3.js* в настоящее время является одним из наиболее популярных инструментов, используемых для графической обработки данных и создания всякого рода диаграмм и графиков.

В отличие от других подобных *JavaScript*-библиотек, *d3.js* не использует *jQuery* для работы со структурой *DOM*, хотя в то же время реализует похожие концепции для работы с объектами.

Для разработки *SVG*-документов с использованием функций библиотеки *d3.js* прежде всего необходимо загрузить саму библиотеку:

```
<script src="http://d3js.org/d3.v3.js"></script>.
```

В табл. 4.7 приведен пример использования простейших функций рисования.

Таблица 4.7

Пример использования библиотеки *d3.js*

Код *HTML*	Рисунок *SVG* и *d3.js*
<html> <head> <script src="http://d3js.org/d3.v3.min.js"> </script> <body> SVG <svg> <rect width="50" height="200" style="fill:red;" /> </svg> D3.js <script> d3.select("body") .append("svg") .append("rect") .attr("width", 50) .attr("height", 200) .style("fill", "blue"); </script> </body> </html>	

Сначала описывается красный прямоугольник с помощью тега *SVG <rect>*. Затем формируется тот же объект с помощью цепочки вызовов функций библиотеки, у которого устанавливаются аналогичные атрибуты, а стиль заполнения *fill* изменяет цвет на синий.

Чтобы понять возможности библиотеки *d3.js*, достаточно посмотреть ряд примеров, которые приводятся разработчиками на [31]. Это прежде всего графики, диаграммы, карты с богатой функциональностью и многое другое.

Для демонстрации особенностей использования библиотеки приведем пример, который строит круговую диаграмму рис. 4.5 (листинг 4.11).

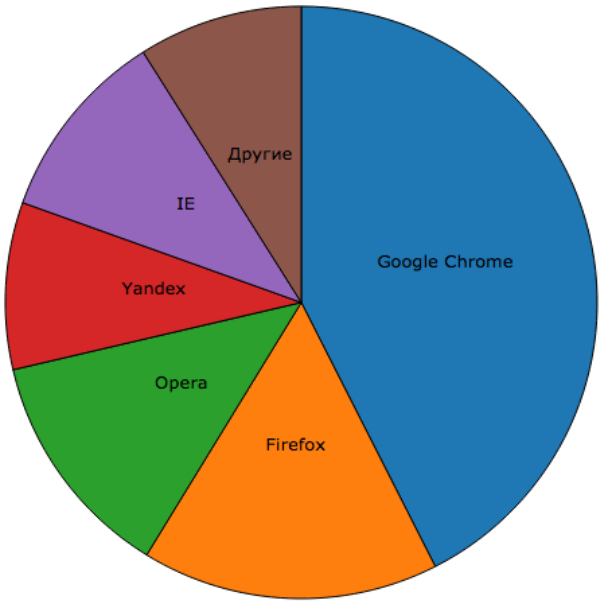


Рис. 4.5. Круговая диаграмма
популярности браузеров

Листинг 4.11. Построение круговой диаграммы:

```
<html>
<head>
<meta charset="utf-8">
<script src="http://d3js.org/d3.v3.min.js"> </script>
<style>
body {font: 13px Verdana;}
.arc path {stroke: #000;}
</style>
<body>
<script type="text/javascript">
var height = 500, width = 500, margin=30,
   data=[{browser: "Google Chrome", rate: 42.52},
        {browser: "Firefox", rate: 16.23},
        {browser: "Opera", rate: 12.6},
        {browser: "Yandex", rate: 9.12},
```

```
        {browser: "IE", rate: 10.56},
        {browser: "Другие", rate: 8.97}];
// функция для получения цветов
var color = d3.scale.category10();
// задаем радиус
var radius = Math.min(width – 2*margin, height- 2*margin) / 2;
// создаем элемент арки с радиусом
var arc = d3.svg.arc()
   .outerRadius(radius)
   .innerRadius(0);
var pie = d3.layout.pie()
   .sort(null)
   .value(function(d) { return d.rate; });
var svg = d3.select("body").append("svg")
       .attr("class", "axis")
       .attr("width", width)
       .attr("height", height)
       .append("g")
       .attr("transform",
          "translate(" +(width / 2) + "," + (height / 2 ) + ")");
var g = svg.selectAll(".arc")
.data(pie(data))
.enter().append("g")
.attr("class", "arc");

g.append("path")
.attr("d", arc)
.style("fill", function(d) { return color(d.data.browser); });

g.append("text")
   .attr("transform", function(d) {
      return "translate(" + arc.centroid(d) + ")"; })
   .style("text-anchor", "middle")
   .text(function(d) { return d.data.browser; });
</script>
</body>
</html>
```

Обработчики событий в *d3.js* добавляются с помощью привычной функции *on()*. В качестве параметров обработчику передается текущий элемент данных и его порядковый номер:

```
.on('click',function(d,i) {concole.log(d,i,d3.event,this);}).
```

Само событие хранится в переменной *d3.event*, а переменная *this* указывает на текущий элемент модели *DOM*.

Полное описание всех функций библиотеки с примерами их использования приводится в [32].

4.5.5. *Библиотека kute.js*

Библиотека включает минимальный набор инструментов с самыми необходимыми функциональными возможностями для веб-разработчиков и дизайнеров, позволяя использовать методы для создания высокопроизводительной, кросс-браузерной анимации. Отличительные черты библиотеки – гибкость, производительность и размер (основные модули 15.8k и 5.6k в сжатом виде). Главное назначение библиотеки – анимация.

Библиотеку можно использовать онлайн, включив в главный файл проекта ссылку:

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/kute.js/1.5.98/kute.min.js"></script>,
```

либо загрузить с ресурса *http://www.jsdelivr.com/projects/kute.js* в директорий проекта. Основной модуль *kute.min.js* позволяет использовать такие методы и свойства, определяющие анимацию, как, например:

- *.to (), .fromTo (), .allTo (), .allFromTo ()*;
- *.start (), .stop (), .pause (), .play ()*;
- scroll: вертикальная прокрутка анимации для окна или любого элемента с переполнением;
- *yoyo, duration, easing, repeat, delay, offset , repeatDelay.*

Кроме того, библиотека имеет еще пять дополнительных модулей, расширяющих ее возможности по анимации:

- *kute-svg* – добавляет морфинг и анимированное рисование;
- *kute-attr* – анимация всех атрибутов;
- *kute-css* – маскирование, расположение фона, размер окон, свойства текста и цвета;
- *kute-text* – анимация счета, написания текста и их комбинация;
- *kute-jquery* – делает скрипт *kute.js* «родным» внутри других *jQuery*-приложений.

Модули также могут вызываться онлайн или загружаться в директорий проекта с ресурса *http://www.jsdelivr.com/projects/kute.js*.

Рассмотрим небольшой пример, демонстрирующий особенности использования библиотеки *kute.js*. Продемонстрируем возможности библиотеки в реализации смешанной трансформации двух квадратов. Создадим директорий проекта *Kute_project* и загрузим туда основной модуль *kute.min.js*. Помимо этого файла разместим здесь главный файл проекта *index.html* (листинг 4.12), рабочий файл с анимацией *tweens.js* (листинг 4.13) и файл стилей оформления *style.css* (листинг 4.14). На рис. 4.6 показаны компоненты проекта *Kute_project*.

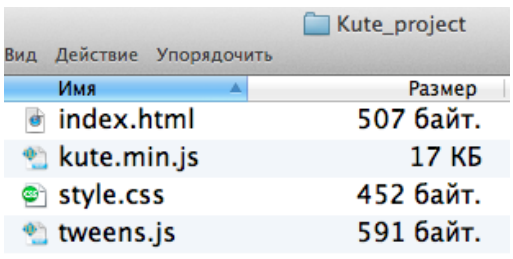


Рис. 4.6. Файлы проекта *Kute_project*

Главный файл обычно именуют *index.html*, и в нем размещается базовая информация об объектах и внешних ресурсах, задействованных в проекте.

Листинг 4.12. Главный файл index.html:

```
<html >
<head>
  <meta charset="UTF-8">
  <title>Смешанная трансформация с KUTE.js</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container" id="container1">
    <div class="box box1">Квадрат №1 размер 150x150</div>
 </div>
  <div class="container" id="container2">
    <div class="box box2">Квадрат №2 размер 150x150</div>
  </div>
<script src='kute.min.js'></script>
<script src="tweens.js"></script>
</body>
</html>
```

В главном файле задаются ссылки на все внешние файлы проекта и определяются два контейнерных класса для двух квадратов.

Листинг 4.13. Файл стилей оформления style.css:

```
body { background-color: #444 }
.container {
    position: relative;
    display: block;}
.box {
    width: 150px; height: 150px;
    position: absolute; top:30px;
    color: #fff;
    font-size: 22px;
    font-weight: bold;
    line-height:45px;
    text-align: center;
    font-family: sans-serif;
    border-radius: 10px;
    background-color: #673AB7;}
.box1 {left: 50px; background-color: #E91E63;}
.box2 {left: 250px; background-color: #FF5722;}
```

В файле стилей определяется цвет фона, размеры обоих квадратов, их смещение по оси *Y*, радиус скругления углов, цвет и параметры текста. Для каждого квадрата отдельно задается смещение по оси *X* и цвет.

Листинг 4.14. Рабочий файл с анимацией tweens.js:

```
// назначение объектам идентификаторов
var el1 = document.querySelector('.box1');
var el2 = document.querySelector('.box2');
// задание анимации
var tween1 = KUTE.fromTo(
 el1,
{translateX:0, rotateX:0,rotateY:0},
{translateX:250,rotateX:360, rotateY:370},
{ duration: 2000, repeat:1, yoyo:true,
easing: 'easingCubicOut'}).start();
var tween2 = KUTE.fromTo(
el2,
{translateX:0,rotateX:0,rotateY:0},
{translateX:-250,rotateX:360, rotateY:370},
{ duration: 2000, repeat:1, yoyo:true,
easing: 'easingCubicOut'}).start();
```

В рабочем файле задается необходимая анимация. Сначала определяются два объекта *el1* и *el2* как экземпляры контейнерных классов, определенных в главном файле проекта. Затем для каждого объекта задается анимация *tween* с помощью метода *fromTo()*: перемещение вдоль оси *X* (*tranlateX*), вращение вдоль осей *X* и *Y* (*rotateX, rotateY*), продолжительность (*duration*), количество повторений (*repeat*), повтор в обратной последовательности (*yoyo*) и закон изменения (*easing*). Наконец, каждая анимация запускается с помощью метода *start()*.

На рис. 4.7 приведены фрагменты результата выполнения анимации, реализованной в проекте *Kut_project*.

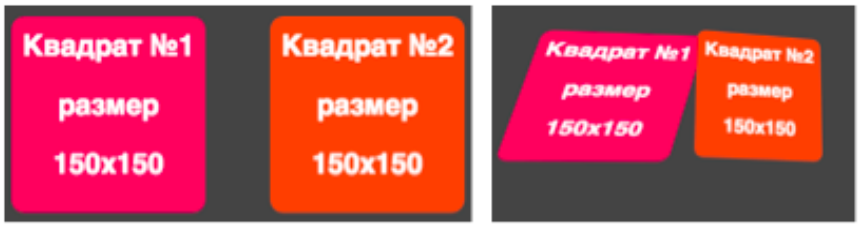


Рис. 4.7. Начальная и промежуточная фазы анимации *Kute_project*

В разд. 3.2.6* упоминалось о том, что данная библиотека предлагает достаточно эффективные методы для реализации мофинга. Действительно, модуль *kute-svg* – добавляет морфинг и анимированное рисование. На сайте *http://thednp.github.io/kute.js/svg.html* можно подробно ознакомиться с приемами данной технологии.

*Филиппов Ф. В. Обработка графической информации в формате SVG : учебное пособие : часть 2 / Ф. В. Филиппов ; СПбГУТ. – СПб., 2017.

5. РЕДАКТОРЫ SVG

Все приложения для создания *SVG*-графики можно разбить на две группы. В первую входят графические пакеты, ориентированные на работу с векторной графикой и позволяющие вместе с тем экспортировать изображения в формат *SVG*. Наиболее популярные из них – *Adobe Illustrator*, *CorelDRAW, AutoCAD* и *Microsoft Visio*. Основным преимуществом данной группы приложений является то, что они обладают расширенным инструментарием для создания векторных изображений и позволяют добиться уникальных эффектов. Однако указанные приложения требуют серьезной специальной подготовки и потому в большей степени рассчитаны на профессиональных дизайнеров. Кроме того, они не ориентированы на *SVG*-файлы, поскольку не допускают корректировки их на текстовом уровне.

Вторую группу программных продуктов образуют пакеты, предназначенные исключительно для создания *SVG*-графики. Они обладают гораздо меньшими возможностями в плане визуальной разработки векторных изображений, хотя и включают весь необходимый инструментарий. Но зато они предоставляют удобные средства для редактирования исходного кода и позволяют работать с *SVG*-объектами как в визуальном режиме, так и на уровне кода, причем между этими вариантами представления информации можно легко переключаться. Все приложения из этой группы достаточно просты и не требуют много времени на освоение. Ниже мы рассмотрим некоторые из них, представляющие наибольший интерес.

На сегодняшний день существует огромное количество редакторов *SVG*, каждый из которых имеет свою область применения. Целью данного раздела является краткое ознакомление с возможностями типичных представителей: от самого простого (*Method Draw*) к широко используемому (*Inkscape*), удобному для первоначального изучения (*Boxy SVG*) и самому новому и перспективному (*Archer Editor).*

5.1. Редактор *Method Draw*

Method Draw – наверное, самый простой редактор для создания и обработки файлов *SVG* [27, 28]. Он является развитием редактора *SVG Edit* с учетом пользовательского опыта использования последнего.

Редактор *Method Draw* является онлайн-приложением, свободно распространяемым по лицензии *MIT*, и загружается непосредственно с адреса [45]. На рис. 5.1 представлен интерфейс редактора.

Конечно, его возможности несколько скромны по отношению к редакторам, рассматриваемым ниже, но доступность иногда делает его незаменимым.

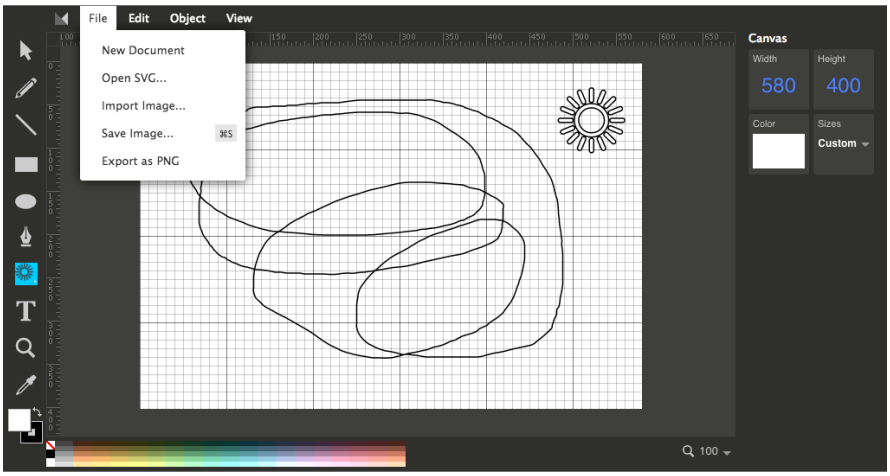


Рис. 5.1. Онлайн-редактор
векторной графики *Method Draw*

В арсенале средств – набор инструментов для встраивания базовых объектов *SVG*, их группировки и трансформации. Из статических визуальных эффектов доступен фильтр *feGaussianBlur*.

Имеется небольшая встроенная библиотека тематических шаблонов, состоящая из десяти категорий *SVG*-объектов. Для любого сформированного графического объекта всегда доступен для просмотра и редактирования SVG-код из меню: *View* → *Source*. Пример оформления фрагмента результирующего кода приведен в листинге 5.1.

Листинг 5.1. Фрагмент кода, формируемого редактором Method Draw:

```
<svg width="580" height="400" xmlns="http://www.w3.org/2000/svg">
 <!-- Created with Method Draw – http://github.com/duopixel/Method-Draw/ -->
<g>
  <title>Layer 1</title>
  <rect id="svg_1" height="44" width="66" y="86" x="128" stroke-width="1.5" stroke="#000" fill="#fff"/>
 </g>
</svg>
```

Как видно из примера, код содержит минимальный набор описательных средств для представления SVG-формата. Стиль представления документа соответствует классической нотации. Результат любого редактирования кода можно тут же наблюдать в визуальном режиме.

5.2. Редактор *Inkscape*

Редактор *Inkscape* – это свободно распространяемое программное приложение с открытым исходным кодом [29]. Он имеет как *WYSIWYG*-интерфейс, существенно облегчающий визуальную разработку, так и интерфейс, позволяющий напрямую управлять кодом *SVG*.

В редакторе *Inkscape* реализована поддержка *A*-каналов, работа со слоями, использование обтекаемого текста, градиентные заливки, большое число фильтров и эффектов, разнообразные трансформации, удобная работа с контурами и объектами, группировка объектов и многое другое. Возможен импорт данных из файлов *JPEG, PNG* и *TIFF* и внедрение их в *SVG*-изображения. На рис. 5.2 представлен интерфейс редактора, который во многом напоминает интерфейс пакета *CorelDRAW.*

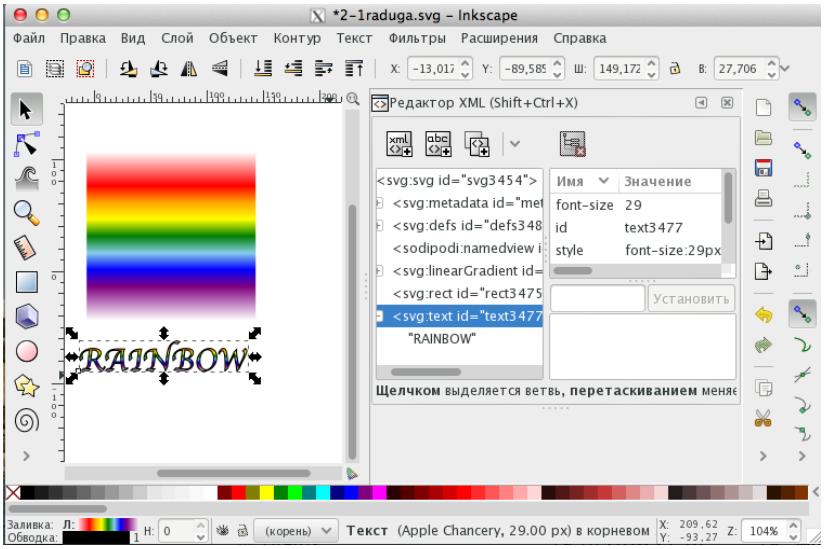


Рис. 5.2. Правка исходного кода
в окне встроенного редактора *Inkscape*

В отличие от *CorelDRAW* пакет *Inkscape* позиционируется как приложение для *SVG*-дизайнеров, поэтому в нем наряду с классическим визуальным созданием графики предусмотрены возможности непосредственной обработки *XML*-кода в среде встроенного текстового редактора.

Сохранение изображений возможно в обычных и сжатых *SVG*-файлах, а при желании их можно конвертировать в распространенные векторные и растровые форматы.

Редактор *Inkscape* отличается удобным русифицированным интерфейсом, прост в освоении и поставляется с неплохой справочной системой, включающей разнообразные примеры использования. Можно констатировать, что данный редактор является одним из самых широко используемых средств разработки *SVG*-документов.

5.3. Редактор *Boxy SVG*

Редактор Boxy SVG – это свободно распространяемое программное приложение. Он является быстрым, простым в использовании и чрезвычайно полезным на практике [30]. В арсенале его средств полный набор инструментов описанных в трех первых разделах данного пособия.

Для повышения производительности работы с этим редактором в нем предусмотрено более 100 команд с помощью настраиваемых горячих клавиш. Его также можно использовать прямо в веб-браузере, при этом импортируя изображения и шрифты из *Google*. На рис. 5.3 представлен интерфейс редактора *Boxy*.

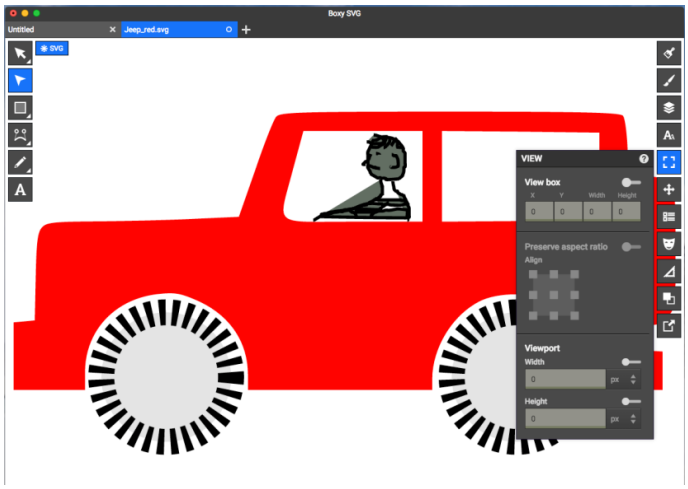


Рис. 5.3. Редактор векторной графики *Boxy*

Панели справа предоставляют меню основных инструментов работы с графикой:

- заливка объекта цветом (*fill*), тип градиента (*type*), способ распространения (*spread*) и прозрачность (*opacity*);
- выбор обводки (*type*), способа распространения (*spread*), цвета (*color*), прозрачности (*opacity*), ширины (*width*), типа конца линии (*line cap*) и способа соединения (*line join*);
- слои (*compositing*) и режим смешивания (*blend mode*);
- текст (*text*), шрифт (*font*), размер (*size*), тип (*bold, Italian etc.*), расстояние между буквами и словами (*letter spacing & word spacing*);
- настройка *viewBox* (*x, y, width, height, preserveAspectRatio*) и *viewport* (*width, height*);
- расположение (*arrangement*), трансформации (*transform*), порядок (*z-order*);
- многоразовые ресурсы – определения (*defs*) и символы (*symbols*);
- маскирование (*mask*), вырезание (*clipping mask*), изменение прозрачности (*opacity mask*);
- геометрия объекта – положение (*x, y*), размер (*width, height*) и поворот (*rotation*);
- путь (*path*), преобразовать в путь (*convert to path*), булевские операции (*Boolean operation: unite, subtract, intersect, exclude*), объединение путей (*compound path*), разделение на части (*break apart*);

• экспорт в формате (*PNG, JPEG*), площадь (*area*), выбрать *viewBox* размер (*size*), прозрачный фон (*transparent background*).

Можно сказать, что это достаточно внушительный список возможностей оформления графики. Наряду с предсказуемыми палитрами цветов и градиентов этот редактор позволяет определять и назначать свои объекты в качестве повторно используемых ресурсов – символов и шаблонов. Это можно сделать, выделив любой объект на холсте и нажав «+» в меню многоразовые ресурсы. При нажатии на «+», когда объект на холсте не выбран, редактор предоставит чистый холст для формирования нового символа.

Одной из замечательных особенностей редактора *Boxy* является то, что он создает аккуратный, эффективный и легко читаемый код *SVG*.

В настоящее время единственным недостатком Boxy является отсутствие возможности применять SVG-фильтры. Тем не менее поскольку имеется доступ к коду, можно с успехом применить ручное кодирования фильтров в готовом SVG-документе.

По большому счету Boxy можно рассматривать не столько как «редактор векторной графики», который формирует SVG-код, но и как WYSIWYG-интерфейс для языка SVG.

5.4. Редактор *Archer Editor*

Данное приложение является одной из последних разработок в области средств проектирования динамической векторной графики на базе SVG. Редактор *Archer Editor* предназначен для реализации больших проектов и позволяет сочетать качество графики SVG и мощь динамики функций *JavaScript*-библиотек [46]. На рис. 5.4 представлен интерфейс редактора *Archer Editor*.

Редактор *Archer Editor* позволяет создавать реактивные векторные графические визуализации для веб-приложения. Процесс разработки состоит из построения *SVG*-бъекта как совокупности графических элементов и указания на то, как должен реагировать тот или иной элемент на определенные внешние воздействия.

Для создания динамической графики используются построенный *SVG*-объект, файл определения поведения графических элементов и *JavaScript*-библиотека *Archer Runtime*. Примеры результатов, полученных с помощью редактора, можно посмотреть в [47]. Редактор *Archer Editor* доступен с *Mac App Store* и *Windows App Store* и является бесплатным для личного и некоммерческого использования.

Структурно редактор *Archer Editor* состоит из трех основных компонентов:

- холста для редактирования и просмотра SVG;
- панели функций преобразования;
- редактора кода.

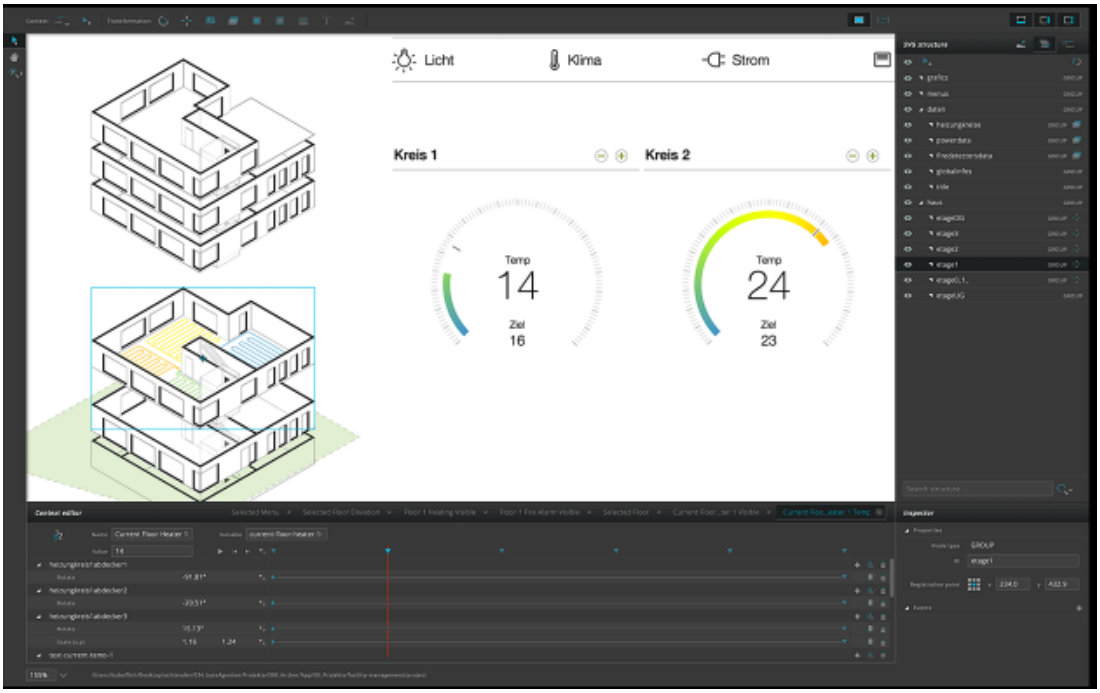


Рис. 5.4. Редактор векторной графики *Archer Editor*

Редактор кода является основой – здесь все собирается вместе: изменяемые переменные, диапазоны значений, SVG-элементы и их преобразования.

Важная особенность редактора *Archer Editor* состоит в том, что он позволяет комбинировать и объединять несколько графических преобразований (поворот, перемещение, масштабирование, изменение цвета и прозрачности) для нескольких элементов в различных диапазонах изменения значений. Он также позволяет динамически изменять как текст, так и демонстрируемые изображения.

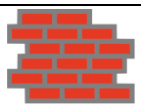
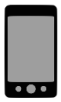
Результирующий код хорошо структурирован и доступен для понимания и ручного редактирования.

6. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

В табл. 6.1 даны задания для самостоятельной работы по разд. 1. Задания из строк D, E выполняются в SVG-редакторе.

Таблица 6.1

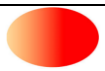
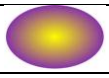
Задания для самостоятельной работы по разд. 1

Задание	1	2	3	4	5
A					
B					
C					
D					
E					

Задания из табл. 6.2 предполагают использование статических визуальных эффектов, описанных в разд. 2.

Таблица 6.2

Задания для самостоятельной работы по разд. 2

Задание	6	7	8	9	10
A					
B					
C					

Задания из табл. 6.3 предполагают использование динамических визуальных эффектов, описанных в разд. 3.

Таблица 6.3

Задания для самостоятельной работы по разд. 3

Задание	11	12	13	14	15
A					

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. *Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation 04 September 2001. – http://www.w3.org/TR/SVG10/.*

2. *Scalable Vector Graphics (SVG) 1.1 (Second Edition), W3C Recommendation 16 August 2011. – http://www.w3.org/TR/SVG/.*

3. *Scalable Vector Graphics (SVG), W3C SVG Working Group. – http://www.w3.org/Graphics/SVG/.*

4. *Scalable Vector Graphics (SVG) 2, W3C Editor's Draft 19 October 2016. – https://svgwg.org/svg2-draft/.*

5. *SVG Working Group specifications. – https://github.com/w3c/svgwg.*

6. *SVG Accessibility API Mappings, W3C Editor's Draft 16 August 2016. – http://w3c.github.io/aria/svg-aam/svg-aam.html.*

7. *https://xmlgraphics.apache.org/batik/Apache™ Batik SVG Toolkit.*

8. *An SVG Primer for Today's Browsers, W3C Working Draft — September 2010. – https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html.*

9. *http://srufaculty.sru.edu/david.dailey/svg/newstuff/simpleshapes.svg.*

10. *Structuring, Grouping, and Referencing in SVG — The <g>, <use>, <defs> and <symbol> Elements – Jule 2014 – перевод: https://habrahabr.ru/post/230443/. – https://sarasoueidan.com/blog/structuring-grouping-referencing-in-svg/.*

11. *http://www.intuit.ru/studies/courses/1063/210/lecture/5424.*

12. *SMIL Animation, W3C Recommendation 04-September-2001. – https://www.w3.org/TR/2001/REC-smil-animation-20010904/.*

13. *A Guide to SVG Animations, December 2015. – https://css-tricks.com/guide-svg-animations-smil/.*

14. *https://svg-art.ru.*

15. *SVG Animations Level 2, W3C Editor's Draft 10 November 2016. – https://svgwg.org/specs/animations/.*

16. *https://developer.mozilla.org/en-US/docs/Web/SVG/Element/defs.*

17. *http://codepen.io/noahblon/post/an-intro-to-svg-animation-with-smil.*

18. *https://www.w3.org/Consortium/Offices/Presentations/SVG/113.svg.*

19. *Coyer, C. SVG symbol a Good Choice for Icons, June 3, 2014. – https://css-tricks.com/svg-symbol-good-choice-icons/.*

20. *SVG Cubic Bézier Curve Example – постоение on-line. – http://blogs.sitepointstatic.com/examples/tech/svg-curves/cubic-curve.html.*

21. *Accessible SVGs, Heather Miglority, July 6, 2016. – https://css-tricks.com/accessible-svgs/.*

22. *https://developer.mozilla.org/ru/docs/Web/SVG.*

23. *http://ruseller.com/lessons.php?rub=28&id=.*

24. Библиотека *svg.js. – http://svgjs.com/.*

25. Библиотека *vivus.js. – https://github.com/maxwellito/vivus#vivusjs.*

26. *http://editor.method.ac он лайн svg редактор Method Draw.*

27. *https://github.com/duopixel/Method-Draw/.*

28. Редактор *Inkscape. – http://www.inkscape.paint-net.ru/.*

29. *Boxy SVG: A Fast, Simple, Insanely Useful, FREE SVG Editor By Alex Walker April 21, 2016. – https://www.sitepoint.com/boxy-svg-a-fast-simple-insanely-useful-svg-editor/.*

30. Библиотека *d3.js. – https://github.com/d3/d3/releases.*

31. Примеры разработок на *d3.js. – http://bl.ocks.org/mbostock, https:// github.com/mbostock/d3/wiki/Gallery.*

32. Описание функций *d3.js. – https://github.com/d3/d3/blob/master/API.md.*

33. Библиотека *vivus.js. – https://github.com/maxwellito/vivus.*

34. Онлайн ресурс. *– https://maxwellito.github.io/vivus-instant/.*

35. Описание библиотеки *svg.js. – http://svgjs.com/.*

36. Синтаксис библиотеки *svg.js. – http://documentup.com/wout/svg.js#syntax-sugar.*

37. Примеры на *svg.js. – https://www.npmjs.com/package/svgjs.*

38. Программная анимация на JavaScript. *– http://javascript.ru/blog/andrej-paranichev/osnovy-programmnoj-animacii-javascript.*

39. Как манипулировать и анимировать *SVG* через *Snap.svg. – http:// htmlbook.ru/blog/kak-manipulirovat-i-animirovat-svg-cherez-snapsvg.*

40. Библиотека *Snap.svg. – https://github.com/adobewebplatform/Snap.svg/ archive/v0.3.0.zip.*

41. Примеры использования библиотеки *Snap.svg. – http://codepen.io/ collection/edpyJ/.*

42. *SVG Tutorial – W3Schools. – https://github.com/svgdotjs/svg.js.*

43. *https://www.youtube.com/watch?v=ehwK-KM4uYQ&feature=youtu.be.*

44. *https://cdnjs.com/.*

45. *http://editor.method.ac.*

46. *http://archer.graphics/home.html.*

47. *http://codepen.io/archer-graphics/.*

Филиппов Феликс Васильевич

ОБРАБОТКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ В ФОРМАТЕ SVG

Учебное пособие

Часть 2

Редактор *И. И. Щенсняк*

Компьютерная верстка *Н. А. Ефремовой*

План издания 2017 г., п. 88 б

Подписано к печати 21.06.2017
Объем 2,25 усл.-печ. л. Тираж 26 экз. Заказ 797

Редакционно-издательский отдел СПбГУТ
191186 СПб., наб. р. Мойки, 61

Отпечатано в СПбГУТ